D0 Downloading

Transferring large blocks of data via 1553

Wed, Sep 10, 1997

The D0 Detector, for its Run II operation, has need of downloading large blocks of data to the detector platform, interfaced via Mil-STD-1553B. This field bus protocol was used heavily during Run I for access to the Rack Monitors (RM's) that provided analog and digital I/O for slow controls. The D0 CDAQ data request software protocol was designed to be used between D0 host programs and the local station front ends. Each 15 Hz cycle, data is acquired from each Remote Terminal (RT) attached to each 1553 controller. Each front end VMEbus crate included either two or four controllers, with a typical load of eight RT's per controller.

For Run II, additional RT's will be added to each controller that will be of a new type that is designed for interfacing to the high speed data acquisition hardware on the platform. Megabytes of data need to be downloaded, and perhaps uploaded for checking, to initialize each experimental data run. Because the amount of data is so large, and because the 1553 bandwidth is 20 µs per word, attention must be paid to the design of software used for downloading. The design goal is to make it possible to keep each controller busy simultaneously, yet to permit the normal 15 Hz data acquisition of RM data to continue.

First, consider the problem of downloading—transferring large blocks of data to various RT's. The normal CDAQ protocol will be used on the host side. Received by a front end, such commands are called "settings." The normal server support vectors to a "set-type routine" depending upon the listype used in the setting message. Upon completion of the setting activity, an acknowledgment reply is returned to the host. The reply includes one word of error status. If any of the setting operations included in the setting message results in failure, this error status word describes the failure, in which case subsequent settings in the message are ignored. The structure of a setting message is an array of listypes, each of which points to an array of idents. For support of this downloading, a new listype will be designed whose ident format indicates the required hardware addressing to be used. The 3-word ident format may be the following:

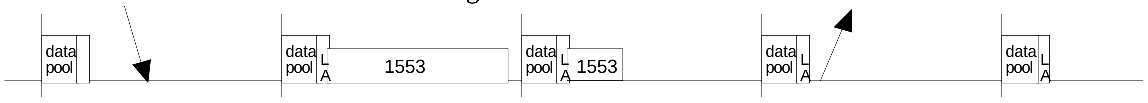
Bits	*Width*	*Function*
47–32:	16	Node number
31–16:	16	Hi word of controller memory base address
15–0:	16	Command word for 1553: RT#, SubAddr#, Mode code.

Besides the ident, there is also the setting data and its stated length. As D0 does not use Internet protocols, this size must necessarily be limited to about 4K bytes to fit in a token ring network frame. To transfer megabytes to an RT, a host program will need to break it down into pieces that fit within this network frame size limit.

The 1553 hardware protocol places a limit of 32 on the number of words moved to or from an RT during execution of one command. In order to transfer more data, multiple 32-word commands must be used. Each such transfer must take about 700 µs to execute at 20 µs per word. (This includes the command word, the 32 data words, up to 14 µs for the RT to delay before replying, and the RT's status word that is returned after accepting the last data word.) In order to maximize efficiency of 1553 command execution, the front end software builds a queue of waiting command blocks in the controller's memory, so that when the end-of-command interrupt occurs, the interrupt routine can immediately present the next queued command to the controller chip, thereby minimizing the controller's idle time. Each controller delivers a separate interrupt, allowing the software to keep multiple controllers busy at the same time.

It is common when adding new functionality to the system software to use local applications (LA's). In this case, a new LA will be designed to manage the construction of the required number of 1553 command blocks to accomplish the download setting activity. It will break down the number of bytes of setting data into 32-word blocks, format the command blocks with the data words in the controller's memory, queue pointers to each command block, and start the controller on the first command. (The controller has DMA access to its own memory, which is why the command blocks must be built therein.) The number of command blocks built is limited by the time remaining in the present 15 Hz cycle. If we assume perfect performance, with 700 µs needed per 32 words, about 100 command blocks should be sufficient for the amount of data that can be transferred by one controller during one cycle. The usual number that will be used is probably less than this, since some time is needed at the start of each 15 Hz cycle to do the normal data pool updating of RM data. The size of each command block will be $50 bytes, which is just large enough for a 32-word transfer. For 100 such blocks, 8K bytes should be enough controller memory space to be used for this purpose. Each controller's memory is 64K bytes. (It is actually 16K bytes at present, but the non-volatile memory chip modules can be upgraded to provide for 64K. Since these chips include a lithium battery with a life of about 10 years, a replacement is probably due.)

The LA is invoked early each 15 Hz cycle, just after the data pool has been updated, so that all 1553 activity is dormant. As the LA recognizes that a requested transfer of all the setting data has finished (during the previous cycle), it must return a reply message to the requesting host. When the D0 protocol task accepts a setting command of this type, it will build a reply message block, but it must refrain from sending it to the host requester as it normally would, until the setting has actually been accomplished and any error status has been checked. So the LA must know the address of this allocated reply message block so it can include the proper error status code and queue the reply to the network. As an example to show the timing of such a command, consider that a setting of 4K bytes has been sent from a host to be transferred to an RT that is connected to a given controller.



The setting message is received during one 66 ms cycle, the first part of the data is transferred during the next cycle, the remaining part is transferred during the third cycle, and the setting acknowledgment reply message is sent back to the host in the fourth cycle. At the start of each cycle is portrayed the data pool updating, followed by the LA's activity, followed by the 1553 controller's activity during execution of the queued command blocks. Only the amount of data that can be scheduled to run during the time remaining in each cycle can be queued to the controller. In this example, two cycles of controller activity were needed; in another case, perhaps only one cycle would be sufficient.

To keep 1553 controller activity busy with a larger duty cycle than the above example illustrates, the host downloading program should send more than one setting without waiting for an acknowledgment; in other words, several setting messages should be outstanding for each controller that is being downloaded. In this way, we can insure that we maximize the usage of each 1553 controller for the time during each cycle that normal data pool updating is not active. This will impact the design of the host downloading program, but it is necessary in order to achieve high downloading efficiency.

In addition to downloading, it will be necessary to support uploading, too. Again, the D0 software protocols will be used. The 1553 commands used will cause the RT to send back data to the controller for deposit in the command blocks. And the received data must then be copied into a reply message block for subsequent delivery to the host. Again, it is desirable to have the LA manage this data acquisition, since the controller can only perform commands to read from or to write to an RT, never both at the same time. To do it in an analogous way to what was done for support of a setting using this new listype, the read-type routine should queue transfer commands for the LA to process. The reply message must be delayed until the data requested has actually been collected. The LA, when it detects that the transfer of data has been completed during the last cycle, must copy the data from the many command blocks into the reply message block. When the last of the requested data has been copied, it should queue the message block for delivery to the host. The host program should maintain a limited number of outstanding requests per local station node so that the communications queue read by the LA does not overflow. There is nohing that can be gained by permitting the host program to get megabytes ahead of the actual downloading that is being done. Indeed, when an error occurs, the host may have to begin a long download again from the beginning, or at least back up to a known successful point of error-free completion.

The form of the communication between the D0 protocol software and the LA may be via a data stream used as a message queue, much as was done for support of Swift Digitizer snapshot waveforms used by Acnet. Use of a data stream may help in preparing diagnostic software to monitor what downloading activities have been performed. The D0 protocol software writes into the queue and the LA reads from the queue, maintaining an "OUT" pointer in the user portion of the data stream queue header.